Operating Systems
ECE344

Kernel Address Space

Ashvin Goel
ECE
University of Toronto
Kernel Address Space

Several options

- OS runs in physical memory
  - Address translation is turned off when in kernel mode
  - OS can access physical memory directly
- OS uses separate virtual address space
  - OS sets up its own page table
- OS mapped to address space of each process
  - OS uses page table of current process to access its memory
Kernel Address Space = Physical Memory

- Virtual address translation is turned off when in kernel mode

- Pros
  - OS can access physical memory directly, including page tables
  - TLB not used, so no need to flush (unless context switch occurs)

- Cons
  - To copy data (e.g., sys call parameters) in and out of user space, OS needs to simulate paging hardware:
    - do address translation in software
    - deal with dirty bit, etc.
  - 32-bit processors can only address 4GB of physical memory
OS Uses Separate Address Space

- **Pros:**
  - Clean design, every process and OS has entire address space

- **Cons:**
  - On system call entry/exit, requires switching MMU context
    - Changing active page tables (i.e., TLB flush) is expensive
  - Copy in/out of system call parameters requires traversing page tables in software + mapping frames into kernel address space

```
P1 invokes system call
  save regs + MMU switch
  system call handler, invoke thread_yield

P2 returns from system call
  restore regs + MMU switch
  system call handler, thread_yield returns

thread switch
```
OS Mapped to Process Address Space

- OS uses current process address space
- Typically, OS is mapped to high addresses in the virtual address space of each process
- If OS executes in the address space of current thread, how does it protect itself? PTE user/kernel bit.
OS Mapped to Thread Address Space

- **Pros:**
  - On system call entry/exit, no MMU switch needed
  - Copy in/out of system call parameters can reuse paging hardware

- **Cons:**
  - Address space of processes is reduced
  - Page table of each process needs to be setup to access OS code
    - Alternatively, hardware may allow OS to bypass page tables, e.g., MIPS

![Diagram](image-url)

- P1 invokes system call
- P2 returns from system call
- save regs
  - system call handler, invoke thread_yield
  - thread switch + MMU switch
- restore regs
  - system call handler, thread_yield returns
Kernel Low Memory
- contiguous in phys. mem
- cannot be paged out

kernel space addresses
0xC0000000

user space addresses
0x00000000

Physical Memory
0x00000000
In which of the three methods discussed in these slides, does the OS have its own page table?
Think Time Answers

- In which of the three methods discussed in these slides, does the OS have its own page table?
  - In Method 2, OS has its own address space, so it has its own page table.
  - In Method 1, OS accesses physical memory directly, so it doesn’t have its own page table. In Method 3, OS reuses the page tables of the processes, so it doesn’t have its own page table.